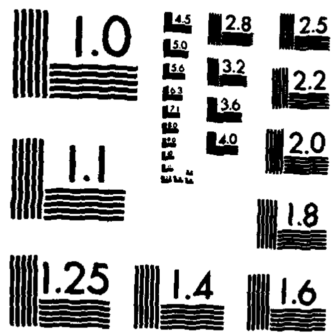
AD-A121 004 LFP USER'S MANUAL (LINCOLN FORTRAN PREPROCESSOR) 1/1
VERSION 0201 FOR IBM/AMD..(U) MASSACHUSETTS INST OF
TECH LEXINGTON LINCOLN LAB J H COSGROVE ET AL.
UNCLASSIFIED 01 SEP 82 TR-623 ESD-TR-82-083 F/G 9/2 NL

END
FILMED
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

[illegible]

Technical Report

623

J.H. Cosgrove
E.T. Bayliss

LFP User's Manual
(Lincoln Fortran Preprocessor)
Version 02.01 for IBM/Amdahl Systems

1 September 1982

Prepared for the Department of the Army
under Electronic Systems Division Contract F19628-80-C-0002 by

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Lexington, Massachusetts

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Department of the Army under Air Force Contract F19628-80-C-0002.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Thomas J. Alpert, Major, USAF
Chief, ESD Lincoln Laboratory Project Office

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

LFP USER'S MANUAL
(LINCOLN FORTRAN PREPROCESSOR)
VERSION 02.01 FOR IBM/AMDAHL SYSTEMS

J.H. COSGROVE
E.T. BAYLISS
Group 47

TECHNICAL REPORT 623

1 SEPTEMBER 1982

Approved for public release; distribution unlimited.

LEXINGTON MASSACHUSETTS

ABSTRACT

LFP (Lincoln Fortran Preprocessor) provides top-down control structures to FORTRAN and generates a self-documenting listing. LFP is compatible with existing FORTRAN and also permits an internal procedure capability.

CONTENTS

LIST OF ILLUSTRATIONS

1.0 INTRODUCTION

The Lincoln Fortran Preprocessor (LFP) Fig. 1-1 was constructed to facilitate structured programming by extending FORTRAN to include the most useful top down control structures. The choice of FORTRAN for a target language was dictated by its being the only higher level language available on many mini-computers. This work was motivated by a desire to make top-down structured programming tools available for the development of FORTRAN software.

LFP is an upward compatible extension of FORTRAN which provides five new top down decision structures, five additional loop structures and an internal procedure capability. In addition to structured control, LFP provides a neat, automatically-formatted, structured listing. The ease of program construction and clarity of program documentation are greatly enhanced thus reducing the clerical detail and the likelihood of programming in bugs.

At Lincoln Laboratory, LFP is implemented on an Amdahl 470 with the CP/CMS operating system, on a Modcomp 4 under MAX 4-rev D operating system and on a PDP 11.

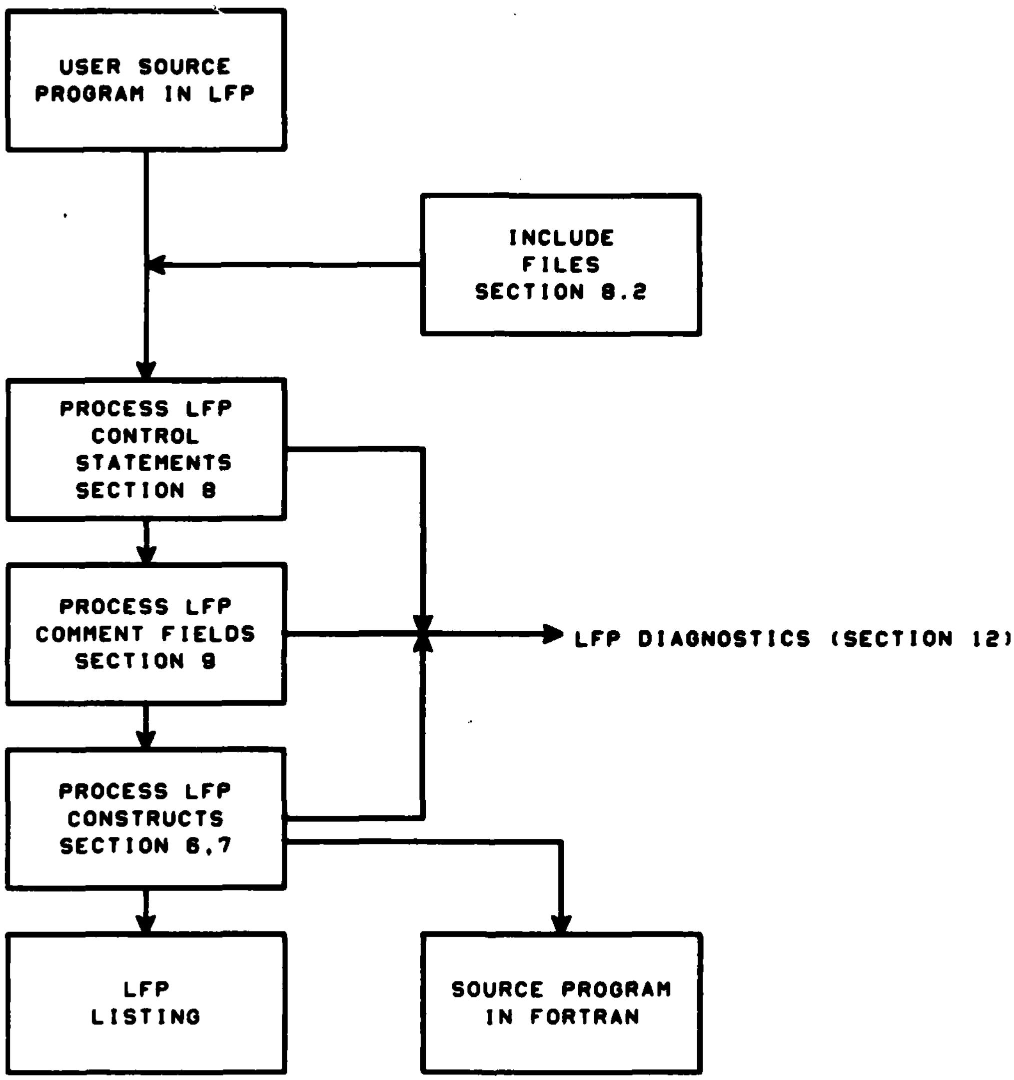


Fig. 1-1. LFP preprocessor.

2.0 RETENTION OF FORTRAN FEATURES

The LFP translator examines each statement in the LFP program to see if it is an extended statement (a statement valid in LFP but not in FORTRAN). If it is recognized as an extended statement, the translator generates the corresponding FORTRAN statements. If, however, the statement is not recognized as an extended statement, the translator assumes it must be a FORTRAN statement and passes it through unaltered. Thus the LFP system does not restrict the use of FORTRAN statements, it simply provides a set of additional statements which may be used. In particular, GO TOs, arithmetic IFs, CALLs, arithmetic statement functions, and any other FORTRAN statements, compiler dependent or otherwise, may be used in LFP programs.

3.0 CORRELATION OF LFP AND FORTRAN SOURCE

A basic flaw in most FORTRAN preprocessors' output is the inability to correlate the preprocessor source listing with compiler syntax or run-time errors. This usually forces the user to list the FORTRAN source that was generated by the preprocessor and to attempt to make sense out of the generally unreadable FORTRAN.

The philosophy inherent in the LFP design was simple: Let LFP work in the same numbering system as the FORTRAN compiler, since all compiler errors or execution errors refer to this numbering system. However, not all compilers number the statements the same way. IBM FORTRAN G and H compilers number every statement except comment and continuation lines while CDC and MODCOMP compilers number every statement. The LFP solution was to allow the user to choose either type of statement numbering.

The statement identification field (line tag) that is present in columns 73 to 80 of the user's source program, if one is present, may at the option of the user be printed on the LFP listing along with the statement. The FORTRAN source generated by LFP may also contain this statement identifier.

It is possible to specify exactly what type of line numbering scheme is to be used by LFP with a LABEL control statement. (See Section 8.1.5.)

A sample of a source program processed by LFP along with the compiler listing output may be found in Section 11.

4.0 STRUCTURED STATEMENTS

A basic notion of LFP is that of the structured statement which consists of a control phrase and its scope. FORTRAN has two structured statements, the logical IF and the DO. The following examples illustrate this terminology:

$$\overbrace{\overbrace{\overbrace{\texttt{IF}}^{\text{keyword}}\quad \overbrace{\texttt{(X.EQ.Y)}}^{\text{specification}}}^{\text{control phrase}}\quad \overbrace{\texttt{U=V+W}}^{\text{scope}}}^{\text{structured statement}}$$

$$\left.\begin{array}{l}\left.\overbrace{\texttt{DO}}^{\text{keyword}}\quad \overbrace{\texttt{30 I = 1,N}}^{\text{specification}}\right\}\text{control phrase}\\ \left.\begin{array}{l}\texttt{A(I) = B(I)+C}\\ \texttt{L(I) = I-K(I)}\\ \texttt{30 CONTINUE}\end{array}\right\}\text{scope}\end{array}\right\}\text{structured statement}$$

Note that each structured statement consists of a control phrase which controls the execution of a set of one or more statements (its scope).

Each control phrase consists of a keyword plus some additional information called the specification. A statement which does not consist of a control phrase and a scope is said to be a simple statement. Examples of simple statements are assignment statements, subroutine CALLs, arithmetic IFs, and GO TOs.

The problem with the Fortran logical IF statement is that its scope may contain only a single simple statement. This restriction is eliminated in the case of the DO, but at the cost of clerical detail (having to stop thinking about the problem while a statement number is invented).

In LFP there is a uniform convention for writing control phrases and indicating their scopes. To write a structured statement, the keyword is placed on a line beginning in column 7 followed by its specification enclosed in parentheses. The remainder of the line is left blank. The statements comprising the scope are placed on successive lines. The end of the scope is indicated by a FIN statement. This creates a multi-line structured statement. Examples of multi-line structured statements:

```
IF (X.EQ.Y)
|  U = V+W
|  R = S+T
└──FIN

DO (I = 1,N)
|  A(I) = B(I)+C
|  C = C*2.14-3.14
└──FIN
```

Note: The statement number has been eliminated from the DO specification since it is no longer necessary, the end of the loop being specified by the FIN.

Nesting of structured statements is permitted to any depth.

Example of nested structured statements:

```
IF (X.EQ.Y)
|  U = V+W
|  DO (I = 1, N)
|  |  A(I) = B(I)+C
|  |  C = C*2.14-3.14
|  └──FIN
|  R = S+T
└──FIN
```

When the scope of a control phrase consists of a single simple statement, it may be placed on the same line as the control phrase and the FIN may be dispensed with. This creates a one-line structured statement. Examples of one-line structured statements:

```
IF (X.EQ.Y) U = V+W
DO (I = 1,N) A(I) = B(I)+C
```

Since each control phrase must begin on a new line, it is not possible to have a one-line structured statement whose scope consists of a structured statement:

Example of invalid construction:

```
IF (X.EQ.Y) DO (I = 1,N) A(I) = B(I)+C
```

To achieve the effect desired above, the IF must be written in a multi-line form.

Example of valid construction:

```
IF (X.EQ.Y)
|  DO (I = 1,N) A(I) = B(I)+C
└──FIN
```

In addition to the IF and DO, LFP provides several useful structured statements not available in FORTRAN. After a brief excursion into the subject of indentation, we will present these additional structures.

5.0 INDENTATION DESCRIPTION

In the examples of multi-line structured statements above, the statements in the scope were indented and an "L" shaped line was drawn connecting the keyword of the control phrase to the matching FIN. The resulting graphic effect helps to reveal the structure of the program. The rules for using indentation and FINs are quite simple and uniform. The control phrase of a multi-line structured statement always causes indentation of the statements that follow its scope. Nothing else causes indentation. A level of indentation (i.e., a scope) can only be terminated with a FIN.

When writing an LFP program on paper, the programmer should adopt the indentation and line drawing conventions shown below. When preparing a LFP source program in machine readable form, however, each statement should begin in column 7. When the LFP translator produces the listing, it will reintroduce the correct indentation and produce the corresponding lines. If the programmer attempts to introduce his own indentation with the use of leading blanks, the program will be translated correctly, but the resulting listing will be improperly indented. The source may be left adjusted to column 7 before processing by the use of the LADJ control card. See Section 8.1.6.

Example of indentation:

1. Program as written on paper by programmer:

```
IF (X.EQ.Y)
    U = V+W
    DO (I = 1,N)
        A(I) = B(I)+C
        C = C*2.14 -3.14
    FIN
    R = S+T
FIN
```

2. Program as entered into computer:

```
IF (X.EQ.Y)
U = V+W
DO (I = 1,N)
A(I) = B(I)+C
C = C*2. 14-3. 14
FIN
R = S+T
FIN
```

3. Program as listed by LFP translator:

```
IF (X.WQ.Y)
.  U = V+W
.  DO (I = 1,N)
.  .  A(I) = B(I)+C
.  .  C = C*2.14-3.14
.  ...FIN
.  R = S+T
...FIN
```

The correctly indented listing is a tremendous aid in reading and working with programs. Except for the dots and spaces used for indentation, the lines are listed exactly as they appear in the source program. That is, the internal spacing of columns 7-72 is preserved. There is seldom any need to refer to a straight listing of the unindented source.

6.0 CONTROL STRUCTURES

The complete set of control structures provided by LFP is described in the following subsections together with their corresponding flow charts. The symbol $\mathcal{L}$ is used to indicate a logical expression. The symbol $\mathcal{S}$ is used to indicate a scope of one or more statements. Some statements, as indicated, do not have a one-line construction.

A convenient summary of the information in this chapter may be found at the end of this section and in Appendix A.

6.1 Decision Structures

Decision structures are structured statements which control the execution of their scopes on the basis of a logical expression or test.

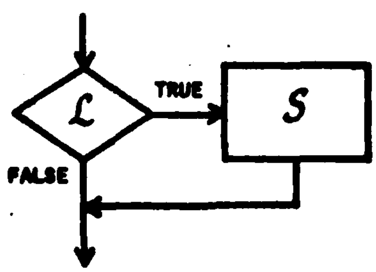
6.1.1 IF

Description: The IF statement causes a logical expression to be evaluated. If the value is true, the scope is executed once and control passes to the next statement. If the value is false, control passes directly to the next statement without execution of the scope.

General Form:

IF ($\mathcal{L}$) $\mathcal{S}$

Flow Chart:



Examples:

```
IF (X.EQ.Y) U = V+W

IF (T.GT.0.AND.S.LT.R)
.  I = I+1
.  Z = 0.1
...FIN
```

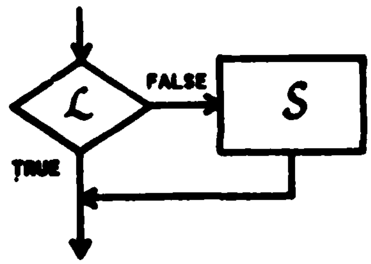
6.1.2 UNLESS

Description: "UNLESS ($\mathcal{L}$)" is functionally equivalent to "IF(.NOT.($\mathcal{L}$))", but is more convenient in some contexts.

General Form:

UNLESS ($\mathcal{L}$) S

Flow Chart:



Examples:

```
UNLESS  (X.NE.Y) U = V+W

UNLESS  (T.LE.O.OR.S.GE.R)
.  I = I+1
.  Z = 0.1
...FIN
```

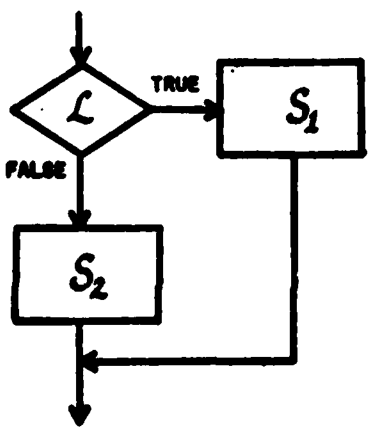
6.1.3 WHEN...ELSE

Description: The WHEN...ELSE statements correspond to the IF...THEN...ELSE statement of Algol, PL/1, Pascal, etc. In LFP, both the WHEN and the ELSE act as structured statements although only the WHEN has a specification. The ELSE statement must immediately follow the scope of the WHEN. The specifier of the WHEN is evaluated and exactly one of the two scopes is executed. *The scope of the WHEN statement* is executed if the expression is true and the scope of the ELSE statement is executed if the expression is false. In either case, control then passes to the next statement following the ELSE scope.

General Form:

WHEN ($\mathcal{L}$) S_1
ELSE S_2

Flow Chart:



Examples:

```
WHEN (X.EQ.Y) U = V+W
ELSE U = V-W

WHEN (X.EQ.Y)
.  U = V+W
.  T = T+1.5
...FIN
ELSE U = V-W
```

```
WHEN (X.EQ.Y) U = V+W
ELSE
.  U = V-W
.  T = T+1.5
...FIN
```

Note: WHEN and ELSE always exist as a pair of statements, never separately. Either the WHEN or the ELSE or both may assume the multi-line form. ELSE is considered to be a control phrase, hence it cannot be placed on the same line as the WHEN. Thus "WHEN ($\mathcal{L}$) S_1 ELSE S_2" is not valid.

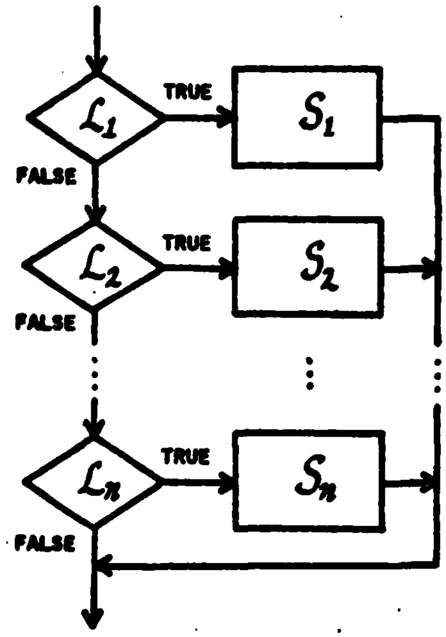
6.1.4 CONDITIONAL

Description: The CONDITIONAL statement is based on the LISP conditional. A list of logical expressions is evaluated one by one until the first expression to be true is encountered. The scope corresponding to that expression is executed, and control then passes to the first statement following the CONDITIONAL. If all expressions are false, no scope is executed. (See, however, the note about OTHERWISE below.)

General Form:

```
CONDITIONAL
. (L1)  S1
. (L2)  S2
.   .    .
.   .    .
.   .    .
. (Ln)  Sn
...FIN
```

Flow Chart:



Examples:

```
CONDITIONAL
.  (X.LT.-5.0)  U = U+W
.  (X.LE.1.0)   U = U+W+Z
.  (X.LE.10.5)  U = U-Z
...FIN
```

```
CONDITIONAL
.  (A.EQ.B)     Z = 1.0
.  (A.LE.C)
.  .  Y = 2.0
.  .  Z = 3.4
.  ...FIN
.  (A.GT.C.AND.A.LT.B)   Z = 6.2
.  (OTHERWISE)   Z = 0.0
...FIN
```

Notes: The CONDITIONAL itself does not possess a one-line form. However, each "($\mathcal{L}_n$) S_n" is treated as a structured statement and may be in one-line or multi-line form.

The reserved word OTHERWISE represents a catchall condition. That is, "(OTHERWISE)S_n" is equivalent to "(.TRUE.)S_n" in a CONDITIONAL statement.

6.1.5 SELECT

Description: The SELECT statement is similar to the CONDITIONAL but is more specialized. It allows an expression to be tested for equality with each expression in a list of expressions. When the first matching expression is encountered, a corresponding scope is executed and the SELECT statement terminates. In the description below, $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$ represent arbitrary but compatible expressions. Any type of expression (integer, real, complex,...) is allowed as long as the underlying Fortran system allows such expressions to be compared with an EQ. OR .NE. operator.

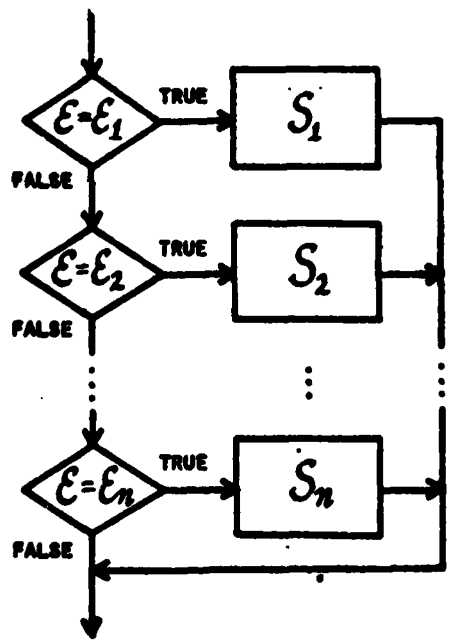
General Form:

SELECT ($\mathcal{E}$)
. ($\mathcal{E}_1$) S_1
. ($\mathcal{E}_2$) S_2
. . . .
. . .
. . .
. ($\mathcal{E}_n$) S_n
...FIN

Example:

```
SELECT (OPCODE(PC))
.  (JUMP)  PC = AD
.  (ADD)
.  .  A = A+B
.  .  PC = PC+1
.  ...FIN
.  (SKIP)  PC = PC+2
.  (STOP)  CALL STOPCD
...FIN
```

Flow Chart:



Notes: As in the case of CONDITIONAL, at most one of the S_i ; will be executed.

The catchall OTHERWISE may also be used in a SELECT statement. Thus "(OTHERWISE)S_n " is equivalent to "($\mathcal{E}$) S_n " within a "SELECT ($\mathcal{E}$)" statement.

The expression is reevaluated for each comparison in the list, thus lengthy, time consuming, or irreproducable expressions should be precomputed, assigned to a variable, and the variable used in the specification portion of the SELECT statement.

6.2 LOOP Structures

The structured statements described below all have a scope which is executed a variable number of times depending on specified conditions.

Of the five loops presented, the most useful are the DO, WHILE, AND REPEAT UNTIL loops. To avoid confusion, the REPEAT WHILE and UNTIL loops should be ignored initially.

6.2.1 DO

Description: The LFPs DO loop is functionally identical to the Fortran DO loop. The only differences are syntactic. In the LFP DO loop, the statement number is omitted from the DO statement, the incrementation parameters are enclosed in parentheses, and the scope is indicated by either the one line or multi-line convention. Since the semantics of the Fortran DO statement vary from one Fortran compiler to another, a flowchart cannot be given. The symbol $\mathcal{I}$ represents any legal incrementation specification.

General Form

DO ($\mathcal{I}$) $\mathcal{S}$

Examples:

```
DO (I = 1,N)  A(I) = 0.0

DO (J = 3,K,3)
.  B(J) = B(J-1)*B(J-2)
.  C(J) = SIN(B(J))
...FIN
```

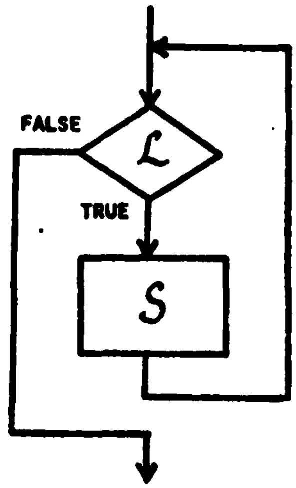
6.2.2 WHILE

Description: The WHILE loop causes its scope to be repeatedly executed while a specified condition is true. The condition is checked prior to the first execution of the scope, thus if the condition is initially false the scope will not be executed at all.

General Form:

WHILE ($\mathcal{L}$) S

Flow Chart:



Examples:

```
WHILE (X.LT.A(I)) I = I+1

WHILE (P.NE.0)
.  VAL(P) = VAL(P)+1
.  P = LINK(P)
...FIN
```

6.2.3 REPEAT WHILE

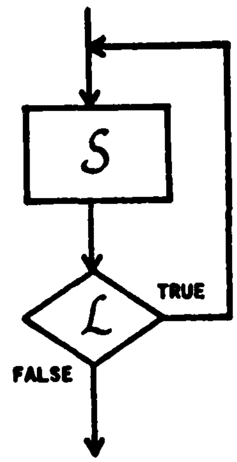
Description: By using the REPEAT verb, the test is logically moved to the end of the loop. The REPEAT WHILE loop caused its scope to be repeatedly executed while a specified condition remains true. The condition is not checked until after the first execution of the scope. Thus the scope will always be executed at least once and the condition indicates under what circumstances the scope is to be repeated.

Note: "REPEAT WHILE ($\mathcal{L}$)" is functionally equivalent to "REPEAT UNTIL (.NOT.($\mathcal{L}$))".

General Form:

REPEAT WHILE ($\mathcal{L}$) S

Flow Chart:



Examples:

```
REPEAT WHILE(N.EQ.M(I)) I = I+1

REPEAT WHILE (LINK(Q).NE.0)
.  R = LINK(Q)
.  LINK(Q)
.. P = Q
.  Q = R
...FIN
```

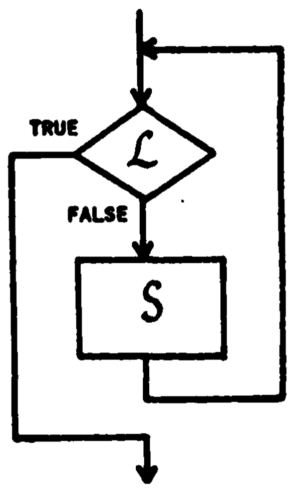
6.2.4 UNTIL

Description: The UNTIL loop causes its scope to be repeatedly executed until a specified condition becomes true. The condition is checked prior to the first execution of the scope, thus if the condition is initially true, the scope will not be executed at all. Note that "UNTIL ($\mathcal{L}$)" is functionally equivalent to "WHILE (.NOT.($\mathcal{L}$)".

General Form:

UNTIL ($\mathcal{L}$) S

Flow Chart:



Examples:

```
UNTIL (X.EQ.A(I))  I = I+1

UNTIL (P.EQ.0)
.  VAL(P) = VAL(P)+1
.  P = LINK(P)
...FIN
```

6.2.5 REPEAT UNTIL

Description: By using the REPEAT verb, the test is logically moved to the end of the loop. The REPEAT UNTIL loop causes its scope to be repeatedly executed until a specified condition becomes true. The condition is not checked until after the first execution of the scope. Thus the scope will always be executed at least once and the condition indicates under what circumstances the repetition of the scope is to be terminated.

General Form:

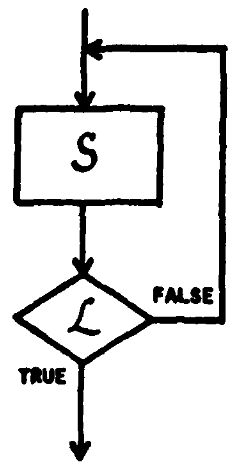
REPEAT UNTIL ($\mathcal{L}$) S

Examples:

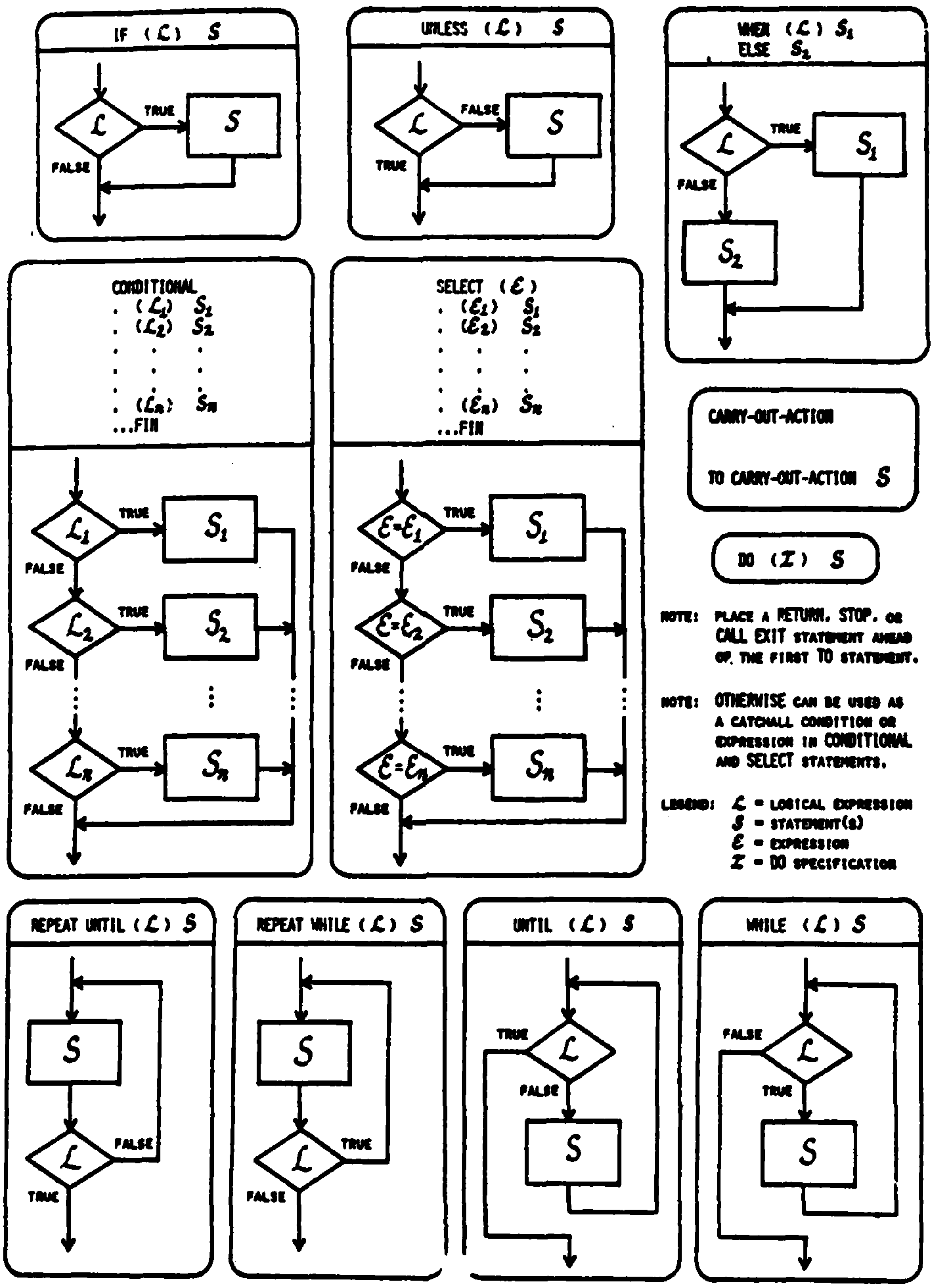
```
REPEAT UNTIL (N.EQ.M(I)) I = I+1

REPEAT UNTIL (LINK(Q).EQ.0)
.  R = LINK(Q)
.  LINK(Q) = P
.  P = Q
.  Q = R
...FIN
```

Flow Chart:



6.3 Control Structure Summary Sheet



7.0 INTERNAL PROCEDURES

In LFP a sequence of statements may be declared an internal procedure and given a name. The procedure may then be invoked from any point in the program by simply giving its name.

Procedure names may be any string of letters, digits, and hyphens (i.e., minus signs) beginning with a letter and containing at least one hyphen. Imbedded blanks are not allowed. The only restriction on the length of a name is that it may not be continued onto a second line.

Examples of valid internal procedure names:

```
INITIALIZE-ARRAYS
GIVE-WARNING
SORT-INTO-DESENDING-ORDER
INITIATE-PHASE-3
```

A procedure declaration consists of the keyword "TO" followed by the procedure name and its scope. The set of statements comprising the procedure is called its scope. If the scope consists of a single simple statement it may be placed on the same line as the "TO" and procedure name, otherwise the statements of the scope are placed on the following lines and terminated with a FIN statement. These rules are analogous with the rules for forming the scope of a structured statement.

General Form of procedure declaration:

TO procedure-name

Examples of procedure declarations:

```
TO  RESET-POINTER  P = 0

TO  DO-NOTHING

TO SUMMARIZE-FILE
.  INITIALIZE-SUMMARY
.  OPEN-FILE
.  REPEAT UNTIL (EOF)
.  . ATTEMPT-TO-READ-RECORD
.  . WHEN (EOF)  CLOSE-FILE
.  . ELSE  UPDATE-SUMMARY
.  ...FIN
.  OUTPUT-SUMMARY
...FIN
```

An internal procedure reference is a procedure name appearing where an executable statement would be expected. In fact, an internal procedure reference is an executable simple statement and thus may be used in the scope of a structured statement as in the last example above. When control reaches a procedure reference during execution of a LFP program, a return address is saved and control is transferred to the first statement in the scope of the procedure. When control reaches the end of the scope, control is transferred back to the statement logically following the procedure reference.

A typical LFP program or subprogram consists of a sequence of Fortran declarations: (e.g. INTEGER, DIMENSION, COMMON, etc.) followed by a sequence of executable statements called the body of the program followed by the LFP internal procedure declarations, if any, and finally the END statement.

Here is a complete LFP program (Fig. 7-1) which illustrates the placement of the procedure declarations.

```
11/13/78   M.I.T. LINCOLN LABORATORY FORTRAN PREPROCESSOR          LFP  02.01
10:07:33                                                           PAGE     1

         %WIDTH 72                                                  PR000010
                  ;INTERATIVE PROGRAM FOR PDP-10                    PR000020
                  ;TO COMPUTE THE SQUARE ROOT OF X.                 PR000030
                  ;STOP WHEN X IS NEGATIVE.                         PR000040
                                                                    PR000050
00001             REAL X,SQRTX                                      PR000060
00002             REPEAT UNTIL ( X .LT. 0.0)                        PR000070
00004             .  READ-IN-A-VALUE-OF-X                           PR000080
00006             .  IF (X .GE. 0.0);ONLY WHEN X IS POSITIVE        PR000090
00007             .  .  COMPUTE-SQRT-OF-X                           PR000100
00009             .  .  TYPE-OUT-THE-RESULT;BOTH X AND SQRTX        PR000110
00011             .  ...FIN                                         PR000120
00012             ...FIN                                            PR000130
00013             STOP;HALT EXECUTION                               PR000140

         ------------------------------------

00014             TO READ-IN-A-VALUE-OF-X                           PR000150
00015             .  TYPE 10                                        PR000160
00016         10  .  FORMAT(' X = ',$)                              PR000170
00017             .  ACCEPT 20,X                                    PR000180
00018         20  .  FORMAT(F);FREE FORMAT INPUT                    PR000190
00019             ...FIN                                            PR000200

         ------------------------------------

00020             TO COMPUTE-SQRT-OF-X   SQRTX=SQRT(X)              PR000210

         ------------------------------------

00023             TO TYPE-OUT-THE-RESULT                            PR000220
00024             .  TYPE 30,X,SQRTX                                PR000230
00025         30  .  FORMAT(' THE SQRT OF ',F7.2,' IS ',F7.2)       PR000240
00026             ...FIN                                            PR000250
00030             END                                               PR000260

         ------------------------------------

                  PROCEDURE CROSS-REFERENCE TABLE

         00020 COMPUTE-SQRT-OF-X
               00007

         00014 READ-IN-A-VALUE-OF-X
               00004

         00023 TYPE-OUT-THE-RESULT
               00009


         NO DIAGNOSTICS

            26 LFP LINES SCANNED,    30 FORTRAN STATEMENTS GENERATED
```

Fig. 7-1. Internal procedure example.

Notes concerning internal procedures:

1. All internal procedure declarations must be placed at the end of the program just prior to the END statement. The appearance of the first "TO" statement terminates the body of the program. The translator expects to see nothing but procedure declarations from that point on.

2. The order of the declarations is not important. Alphabetical by name is an excellent order for programs with a large number of procedures.

3. Procedure declarations may not be nested. In other words, the scope of a procedure may not contain a procedure declaration. It may of course contain executable procedure references.

4. Any procedure may contain references to any other procedures (excluding itself).

5. Dynamic recursion of procedure referencing is not permitted.

6. All program variables within a main or subprogram are global and are accessable to the statements in all procedures declared within that same main or subprogram.

7. There is no formal mechanism for defining or passing parameters to an internal procedure. When parameter passing is needed, the Fortran function or subroutine subprogram mechanism may be used or the programmer may invent his own parameter passing methods using the global nature of variables over internal procedures.

8. The LFP translator separates procedure declarations on the listing by dashed lines as shown in the preceding example.

9. Internal procedure references called from inside nested DO Loops is not recommended.

8.0 CONTROL STATEMENTS

Statements which supply information to the LFP translator during processing are called control statements. These statements, denoted by a control character in column 1, allow user control over the format (appearance) of the LFP listing and permit the inclusion of the contents of other files in the source.

A control statement, in general, will contain 3 items:

control character	This is a percent sign (%) in column 1. This character may be changed by a CONTROL control statement. See section 8.1.2.
control word	This is a string from 1 to 8 characters in length that denotes the control card type and must not contain imbedded blanks.
argument	This is either a numeric or alphabetic string (optional for some keywords).

The only requirement on the control statement format is that the control word comes before the argument and that they are separated by at least one blank. Otherwise, the control word and argument are typed in a field-free format.

Each control word may be recognized from a subset of the complete control word, e.g., the control word INCLUDE can be recognized by an I, IN,..., or INCLUDE. The minimum recognition pattern is denoted by the capital letters in each control word.

All statements will be listed in the LFP listing except those that follow a NOLIST control. A subsequent LIST control statement will negate the effect of a NOLIST.

Control statements can occur anywhere in the user's LFP source file or in the included files.

8.1 Listing Format Controls

The format control statements control the appearance of the LFP listing such as page width, spacing and page length, etc.

8.1.1 Comment delimiter   % COMment char   Default ;

This statement defines the comment field delimiter character, which should not be part of the standard Fortran character set for obvious reasons. A comment field may be placed on any LFP source statement including control statements. The delimiter does not have to be separated from the LFP statement by a blank.

Examples:

```
%COM $
%       COMMENT      ; $ change delimiter back to a ;
```

The first example changes the current comment delimiter to a $. Example 2 then changes the delimiter back to a ; (the field "$ change delimiter back to a ;" is treated as a comment).

8.1.2 Control character   % Control chars   Default %

This control statement allows the user to define a character or a set of characters that will identify a control statement. Caution - Do not use the character C or regular comments will be flagged as BAD CONTROL CARDS.

Unrelated examples:

```
%C  *  ; change control character to a *
%  CON  +-AX
%CONTROL   ;+
```

In example 2 any statement with a +, -, A, or X in column 1 is treated as a control statement. In example 3 the control character is set to , the field ";+" is an inline comment.

If the argument is not present the default control character is assumed.

8.1.3 Double Spacing   % DS

This control statement will initiate double spacing on the LFP listing. Errors are still single spaced. Double spacing is done by carriage control.

8.1.4 Heading   % Heading character string

This control statement defines a character string that will print as heading information at the top of each output page of the LFP listing. The string is not delimited by single quotes and may contain imbedded blanks up to a length of 69 characters.

If the heading length is larger than the page width, the heading is truncated on the right.

The default Heading is

M.I.T. LINCOLN LABORATORY FORTRAN PREPROCESSOR

Examples:

```
%H SUBROUTINE RENAME
%HEADING SYSTEM RS2-TEST
%HEAD  INS SIMULATION;MODEL 4
```

In the third example the field ";MODEL 4" is treated as a comment and will not be part of the heading.

8.1.5 Statement Numbering   % LABEL XYZ

The LABEL control statement determines the method of statement numbering on the LFP listing. The user is presented with the following choices:

1. *FORTRAN line numbers.* These are internally generated by the compiler and are affixed to the listing output to the left of the statements. They are not to be confused with statement labels (found in columns 1 - 5). FORTRAN line numbers are sequential from card to card, however, some FORTRAN compilers, e.g., IBM, do not number comments or continuations.

2. LFP line numbers. These are internally generated by LFP and are sequential from statement to statement.

3. Line tags. These are the 8 column identification field found in columns 73 - 80 of the input LFP statements.

In the LFP listing of a sample program in Section 11, the numbers to the left of the statements are FORTRAN line numbers while those on the right side are line tags. These were chosen by the options available for the XYZ argument field on the LABEL control statement.

Field	Value	Description
X	0	Increment by 1 the FORTRAN line number for every line of FORTRAN generated. This corresponds to most CDC and MODCOMP compilers.
	1	Increment by 1 the FORTRAN line number for every line of FORTRAN generated except comments or continuations. This is standard for IBM machines. DEFAULT.
Y	0	Put LFP line numbers to the left of the source statements.
	1	Put the FORTRAN line numbers as determined by the field X to the left of the source. DEFAULT.
Z	0	Put LFP line numbers to the right of the source.
	1	Put the 8 column line tag to the right of the source. DEFAULT.
	2	Have no field to the right of the source.

The sample LFP listing in Section 11 was prepared with the default LABEL control statement.

Examples:

```
%LABEL 111    ; This is the default
%LABEL        ; This resets the default to XYZ=111
%LABEL 12     ; X=0,Y=1,Z=2
```

8.1.6 Left Adjust % LAdj

This control statement will left adjust the input source to column 7, i.e., all blanks from column 7 to the first nonblank character of each source statement will be removed. This is particularly useful if the source had been indented on input, because LFP does its own indenting.

8.1.7 # Lines/Page % Line N

This statement specifies the number of printed lines on each page of the LFP listing which includes 3 lines for the heading. Reasonable values for N are between 50 and 60 for a line printer. For a terminal with a roll of paper, N could be made very large which would prevent the top of page headers from being written. The default is 60 lines/page.

Examples:

```
%L  55
%  LINE 60
```

8.1.8 Listing Control % LIST

This control statement generates the LFP listing. This may be used in conjunction with the %NOLIST control to selectively list portions of the program. Initially the %LIST control is in effect.

8.1.9 No Left Adjust % NOLAdj

This control negates the effect of the LADJ control, i.e., do not left adjust the LFP source. NOLADJ is the default.

8.1.10 No Listing % NOList

The control turns off the generation of the LFP listing. Only the presence of a LIST card will turn the listing back on. A NOLIST statement is printed except if it is the first record of the source file being processed.

8.1.11 Page Eject % Page N

This control statement forces a page eject on the LFP listing if N is zero or missing. If N is positive, this statement acts as a conditional page eject to keep blocks of the listing contiguous. If there are fewer than N lines left on the page, then eject a page.

If the LFP listing is double or triple spaced (see DS and TS controls) the N means double or triple spaced lines.

Examples:

```
%P
%  Page 20
```

8.1.12 Single Spacing % SS

This control will single space the LFP listing, which is the default spacing.

8.1.13 Triple Spacing % TS

This control statement will triple space the LFP listing.

Note: This is done by carriage control, not inserting blank lines.

8.1.14 Listing Width % Width N

The width control statement specifies the page width of the LFP listing in characters. This affects all output - including page headers and the procedure cross reference table.

N will usually range from 72 to 133 with 1 column being reserved for carriage control. See the sample LFP listing in Section 11 with a column width of 78. 17 columns are dedicated for statement numbering and labelling leaving N-17 columns for the indented source statements. The default page width is 110 columns.

Examples:

```
%W  133
%WIDTH 80; SET WIDTH TO 80 COLUMNS
```

8.2 Inclusion of External Files

8.2.1 INCLUDE Command   % Include FILENAME FILETYPE

The INCLUDE control statement allows the user to include in the source program the contents of other files. This is particularly effective in the usage of common blocks.

For example a series of common definitions are put in a file named COMMON. The user's source program would contain a %INCLUDE COMMON statement to include the common definitions. Included files may contain nested %INCLUDE statements up to a level of 40, however, recursive includes are not allowed.

The filename must be a legitimate CMS filename. The default filetype is LFP.

Examples:

```
%I  CBLOCK1
%  INCLUDE  ABLOC
%  INCLUDE  DEFIN  FORTRAN
```

If the filename is missing on the INCLUDE card or if the file does not exist, the statement is ignored with diagnostic being issued to the LFP listing and the terminal.

8.2.2 Include Expansion   % INCExp N

This control statement controls the expansion of the %INCLUDE file. If N is 1 the file is included, if N is 0 the file is not included in either the LFP listing or the generated Fortran. The default for N is 1.

Examples:

```
%INCEXP 0
%  INCE 1
```

8.3 Control Statement Summary

	Section
%COMment X	8.1.1
Define the comment delimiter character X. Default is ;	
%Control X	8.1.2
Define the control character X. Default is %	
%DS	8.1.3
Double space the LFP listing	
%Heading CHARACTER STRING	8.1.4
Define the heading CHARACTER STRING to go at the top of each page of the LFP listing.	
%Include FILENAME	8.2.1
Include the contents of the file = FILENAME FILETYPE in the source file. The filetype must be LFP.	
%INCExp N	8.2.2
Controls the inclusion of a file on the INCLUDE card. if N is 0 the file is not included, if N is 1 the file is included. Default = 1.	
%LABEL XYZ	8.1.5
X 0 Modcomp FORTRAN line numbering 1 Amdahl FORTRAN line numbering (default) Y 0 LFP line numbers at left of listing 1 FORTRAN line numbers at left of listing (default) Z 0 LFP line numbers at right of listing 1 Line tags at right of listing (default) 2 blanks	
%LADJ	8.1.6
Left adjust the source to column 7, removing blanks.	
%Line N	8.1.7
Print N lines per page. Default = 60.	
%List	8.1.8
Print LFP listing. Default.	
%NOLadj	8.1.9
Do not left adjust source to column 7 (default).	
%NOList	8.1.10
Turn off LFP listing.	
%Page N	8.1.11
Eject a page if N=0 or N is missing. Eject a page if there are less than N lines left on a page.	
%SS	8.1.12
Single space LFP listing (default).	
%TS	8.1.13
Triple space LFP listing.	
%Width N	8.1.14
Width of LFP output listing in characters. Default = 110	

8.4 Control Statement Example - Typical Program Setup.

The following control statements at the beginning of each source program generate a listing that greatly facilitates referencing.

```
%NOLIST
%HEADING SUBROUTINE NAME
%PAGE
%LIST
     SUBROUTINE NAME
     .
     .
     .
     .
     END

%NOLIST
%HEADING SUBROUTINE N2
%PAGE
%LIST
     SUBROUTINE N2
     .
     .
     .
     END
```

9.0 COMMENTS

Comments in LFP are recognized by the presence of a specified comment delimiter in any column or by the traditional method of the character "C" in column 1. All characters to the right of and including the delimiter are considered the comment field.

Comments can be isolated, that is, the source statement is only a comment, or they can be inline, meaning a statement and a comment field may be present on the same source line.

All source lines of LFP including control statements may contain inline comments. There does not have to be a blank between the last character of the statement and the comment delimiter.

Isolated comments are indented to the current LFP listing level if columns 2-6 of the statement are blank. An inline comment is indented only if the statement is indented.

Inline comment fields are stripped off the input statements before the Fortran output is produced. No comments are sent to the generated Fortran.

Examples:

```
C     NORMAL COMMENT
C THIS COMMENT WILL NOT BE INDENTED
;     THE SEMICOLON IS THE DEFAULT DELIMETER
          ; THE DELIMETER MAY BE IN ANY COLUMN
      A = SQRT(B*B+C*C); COMPUTE RADIUS OF CIRCLE
%COMMENT $  ; CHANGE DELIMETER TO  A  $
      DETERMINE-NEXT-EVENT$BY A TABLE LOOKUP
```

10.0 RESTRICTIONS AND CONVENTIONS

If LFP were implemented by a nice intelligent compiler this section would be much shorter. Thus the LFP programmer must observe the following restrictions.

1. LFP must invent many statement numbers in creating the FORTRAN program. It does so by beginning with a large number (usually 99999) and generating successively smaller numbers as it needs them. Do not use a number which will be generated by the translator. A good rule of thumb is to avoid using 5 digit statement numbers.

2. LFP must generate integer variable names. It does so by using names of the form "Innnnn" when nnnnn is a 5 digit number related to a generated statement number. Do not use variables of the form Innnnn and avoid causing them to be declared other than INTEGER. For example the declaration "IMPLICIT REAL (A-Z)" leads to trouble. Try "IMPLICIT REAL (A-H, J-Z) instead.

3. The preprocessor does not recognize continuation lines in the source file. Thus FORTRAN statements may be continued since the statement and its continuations will be passed through the preprocessor without alteration. (See chapter 2.) However, an <u>extended LFP statement which requires translation may not be continued</u>. The reasons one might wish to continue a LFP statement are 1) It is a structured statement or procedure declaration with a one statement scope too long to fit on a line, or 2) it contains an excessively long specification portion or 3) both of the above. Problem 1) can be avoided by going to the multi-line form. Frequently problem 2) can be avoided when the specification is an expression (logical or otherwise) by assigning the expression to a variable in a preceding statement and then using the variable as the specification. Avoid continued IF statements.

4. Blanks are meaningful separators in LFP statements: don't put them in dumb places like the middle of identifiers or key words and do use them to separate distinct words like REPEAT and UNTIL.

5. Let LFP indent the listing. Start all statements in col. 7 and the listing will always reveal the true structure of the program. (as understood by the preprocessor of course). The control statement %LADJ allows for preindented source code.

6. As far as the preprocessor is concerned, FORMAT statements are executable FORTRAN statements since it doesn't recognize them as extended LFP statements. Thus, only place FORMAT statements where an executable FORTRAN statement would be acceptable. Don't put them between the end of a WHEN statement and the beginning of an ELSE statement. Don't put them between procedure declarations.

Incorrect Examples:

```
   WHEN (FLAG)  WRITE(3,30)
30 FORMAT(7H TITLE:)
   ELSE  LINE = LINE+1
```

Corrected Examples:

```
   WHEN (FLAG)
   .  WRITE(3,30)
30 .  FORMAT(7H TITLE:)
   ...FIN
   ELSE  LINE = LINE+1
```

Incorrect Examples:

```
   TO WRITE-HEADER
   .  PAGE = PAGE+1
   .  WRITE(3,40) H,PAGE
   ...FIN
40 FORMAT (70A1,I3)
```

Corrected Examples:

```
   TO WRITE-HEADER
   .  PAGE = PAGE+1
   .  WRITE(3,40) H, PAGE
40 .  FORMAT(70A1,I3)
   ...FIN
```

7. The preprocessor recognizes extended LFP statements by the process of scanning the first identifier on the line. If the identifier is one of the LFP keywords IF, WHEN, UNLESS, FIN, etc., the line is assumed to be a LFP statement and is treated as such. Thus, the LFP keywords are reserved and may not be used as variable names. In case of necessity, a variable name, say WHEN, may be slipped past the preprocessor by embedding a blank within it. Thus "WH EN" will look like "WH" followed by "EN" to the preprocessor which is blank sensitive, but line "WHEN" to the compiler which ignores blanks.

8. In scanning a parenthesized specification, the preprocessor scans from left to right to find the parenthesis which matches the initial left parenthesis of the specification. The preprocessor, however, is ignorant of Fortran syntax including the concept of Hollerith constants and will treat Hollerith parenthesis as syntactic parenthesis. Thus, avoid placing Hollerith constants containing unbalanced parenthesis within specifications. If necessary, assign such constants to a variable, using a DATA or assignment statement, and place the variable in the specification.

Incorrect Example:

```
IF (J.EQ.'(')
```

Corrected Example:

```
LP = '('
IF(J.EQ.LP)
```

9. LFP will not supply the statements necessary to cause appropriate termination of main and sub-programs. Thus it is necessary to include the appropriate RETURN, STOP, or CALL EXIT statement prior to the first internal procedure declaration. Failure to do so will result in control entering the scope of the first procedure after leaving the body of the program. Do not place such statements between the procedure declarations and the END statement.

10. LFP ignores blank lines and does not pass comments or blank lines on to the compiler. Thus blank lines can be used for program clarity without worry.

11. Some FORTRAN compilers allow branching in and out of DO LOOPS-other compilers prohibit this. The usage of internal procedure references inside DO structures is not recommended.

11.0 EXAMPLE OF LFP LISTING

The user's program is named QDROOT LFP which is listed in Fig. 11-2.

Figure 11-1 is the resulting LFP listing. Note the correlation between the line numbers on the left of the LFP listing with the lines on the Fortran compiler output (Figure 11-3). This is accomplished by the LABEL control card (Section 8.1.5).

```
05/14/81      SUBROUTINE QDROOT      SOLVE QUADRATIC FORMULA      LFP  02.01
16:11:19                                                          PAGE     1

      %LIST
00001        SUBROUTINE QDROOT(A,B,C,X1,X2,IERR)

00002        REAL*8 A,B,C,X1,X2,DISCRM,TERM1,TERM2

            ;SOLVE QUADRATIC EQUATION .... A*X*X + B*X + C = 0
            ;FOR X1 AND X2.
            ;IERR IS AN ERROR CODE
            ;       -2      A AND B ARE 0.0---NO ROOTS
            ;       -1      DISCRIMINANT < = 0.0---IMAGINARY ROOTS
            ;        0      NORMAL RETURN---2 REAL ROOTS
            ;        1      HIGH ORDER COEFFICIENT ,A, IS ZERO

00003        IERR=0
00004        WHEN (A .NE. 0.0D0)
00005        .  DISCRM=B*B-4.0D0*A*C;  CALCULATE THE DISCRIMINANT
00006        .  WHEN (DISCRM .LT. 0.0D0)
00007        .  .  IERR=-1;     SET NEGATIVE DISCRIMINANT CODE
00008        .  .  X1=0.0D0
00009        .  .  X2=0.0D0
00010        .  ...FIN
00011        .  ELSE
00011        .  .  TERM1=-B/(2.0D0*A)
00012        .  .  TERM2=DSQRT(DISCRM)/(2.0D0*A)
00013        .  .  X1=TERM1+TERM2;      CALCULATE ROOTS X1 and X2
00014        .  .  X2=TERM1-TERM2
00015        .  ...FIN
00015        ...FIN
00016        ELSE;     THE HIGH ORDER COEFFICIENT IS ZERO
00016        .  WHEN (B.NE.0.0D0)
00017        .  .  X1=-C/B
00018        .  .  X2=X1
00019        .  .  IERR=1
00020        .  ...FIN; WHEN
00021        .  ELSE; A AND B ARE BOTH 0.0
00021        .  .  X1=0.0D0
00022        .  .  X2=0.0D0
00023        .  .  IERR=-2
00024        .  ...FIN; ELSE
00024        ...FIN
00025        RETURN
00026        END

    NO DIAGNOSTICS
      46 LFP LINES SCANNED,     26 FORTRAN STATEMENTS GENERATED
```

Fig. 11-1. LFP listing of user's program.

```
FILE:  QDROOT  LFP    A      5/14/81 16:09      M.I.T. LINCOLN LABORATORY

%NOLIST
%HEAD SUBROUTINE QDROOT           SOLVE QUADRATIC FORMULA
%WIDTH 78
%PAGE
%LIST
          SUBROUTINE QDROOT(A,B,C,X1,X2,IERR)

          REAL*8 A,B,C,X1,X2,DISCRM,TERM1,TERM2

          ;SOLVE QUADRATIC EQUATION .... A*X*X + B*X + C = 0
          ;FOR X1 AND X2
          ;IERR IS AN ERROR CODE
          ;       -2     A AND B ARE 0.0---NO ROOTS
          ;       -1     DISCRIMINANT < = 0.0 ---IMAGINARY ROOTS
          ;        0     NORMAL RETURN---2 REAL ROOTS
          ;        1     HIGH ORDER COEFFICIENT ,A, IS ZERO

          IERR=0
          WHEN (A .NE. 0.0D0)
          DISCRM=B*B-4.0D0*A*C;     CALCULATE THE DISCRIMINANT
          WHEN (DISCRM .LT. 0.0)
          IERR=-1;      SET NEGATIVE DISCRIMINANT CODE
          X1=0.0D0
          X2=0.0D0
          FIN
          ELSE
          TERM1=-B/(2.0D0*A)
          TERM2=DSQRT(DISCRM)/(2.0D0*A)
          X1=TERM1+TERM2;      CALCULATE ROOTS X1 AND X2
          X2=TERM1-TERM2
          FIN
          FIN
          ELSE;     THE HIGH ORDER COEFFICIENT IS ZERO
          WHEN (B.NE.0.0D0)
          X1=-C/B
          X2=X1
          IERR=1
          FIN; WHEN
          ELSE; A AND B ARE BOTH 0.0
          X1=0.0D0
          X2=0.0D0
          IERR=-2
          FIN; ELSE
          FIN
          RETURN
          END
```

Fig. 11-2. User's source program.

```
FORTRAN IV G1  RELEASE 2.0        QDROOT                    DATE = 81134

             SUBROUTINE QDROOT(A,B,C,X1,X2,IERR)                       QDR00060
             REAL*8 A,B,C,X1,X2,DISCRM,TERM1,TERM2                     QDR00080
             IERR=0                                                    QDR00180
             IF(.NOT.(A .NE. 0.0D0)) GO TO 99998                       QDR00190
             DISCRM=B*B-4.0D0*A*C                                      QDR00200
             IF(.NOT.(DISCRM .LT. 0.0)) GO TO 99996                    QDR00210
             IERR=-1                                                   QDR00220
             X1=0.0D0                                                  QDR00230
             X2=0.0D0                                                  QDR00240
             GO TO 99997                                               QDR00250
       99996 TERM1=-B/(2.0D0*A)                                        QDR00270
             TERM2=DSQRT(DISCRM)/(2.0D0*A)                             QDR00280
             X1=TERM1+TERM2                                            QDR00290
             X2=TERM1-TERM2                                            QDR00300
       99997 GO TO 99999                                               QDR00320
       99998 IF(.NOT.(B.NE.0.0D0)) GO TO 99994                         QDR00340
             X1= -C/B                                                  QDR00350
             X2=X1                                                     QDR00360
             IERR=1                                                    QDR00370
             GO TO 99995                                               QDR00380
       99994 X1=0.0D0                                                  QDR00400
             X2=0.0D0                                                  QDR00410
             IERR=-2                                                   QDR00420
       99995 CONTINUE                                                  QDR00440
       99999 RETURN                                                    QDR00450
             END                                                       QDR00460
```

Fig. 11-3. FORTRAN listing of user's program.

```
05/14/81            SUBROUTINE QDROOT(A,B,C,X1,X2,IERR)           XREF 02.02
16:12:20                                                          PAGE     1

SYMBOL      - - - - - - - - -  REFERENCES   - - - - - - - - - -

99994   -   16     21*
99995   -   20     24*
99996   -    6     11*
99997   -   10     15*
99998   -    4     16*
99999   -   15     25*
A       -    1AG    2RL    4      5      11     12
B       -    1AG    2RL    5     11      16     17
C       -    1AG    2RL    5     17
DISCRM  -    2RL    5=     6     12
DSQRT   -   12
IERR    -    1AG    3=     7=    19=     23=
QDROOT  -    1SN
RETURN  -   25RE
TERM1   -    2RL   11=    13     14
TERM2   -    2RL   12=    13     14
X1      -   1AG    2RL     8=    13=     17=    18     21=
X2      -   1AG    2RL     9=    14=     18=    22=

THE FOLLOWING SYMBOLS UNDEFINED OR FUNCTION CALLS
DSQRT   -   12

          END OF XREF PROCESSING
```

Fig. 11-4. Variable cross reference listing.

12.0 ERRORS

This section provides a framework for understanding the error handling mechanisms of version 02.01 of the LFP preprocessor. After each execution of LFP the message NO DIAGNOSTICS is sent to the terminal and the listing if there were no errors. If there were errors the message ERRORS - MAJOR xxxxx, MINOR yyyyy, CONTROL CARDS zzzzz is printed.

LFP examines an LFP program on a line by line basis. As each line is encountered it is first subjected to a limited syntax analysis followed by a context analysis. Errors may be detected during either of these analyses. It is also possible for errors to go undetected by the preprocessor.

12.1 Syntax Errors

The fact that a statement has been ignored may, of course, cause some context errors in later statements. For example the control phrase "WHEN (X(I).LT.(3+4)" has a missing right parenthesis. This statement will be ignored, causing as a minimum the following ELSE to be out of context. The programmer should of course be aware of such effects. More is said about them in the next section.

12.2 Context Errors

If a statement successfully passes the syntax analysis, it is checked to see if it is in the appropriate context within the program. For example, an ELSE must appear following a WHEN and nowhere else. If an ELSE does not appear at the appropriate point or if it appears at some other point, then a context error has occurred. A frequent source of context errors in the initial stages of development of a program comes from miscounting the number of FIN's needed at some point in the program.

With the exception of excess FIN's which do not match any preceding control phrase and are ignored all context errors are treated with a uniform strategy. When an out-of-context source statement is encountered, the translator generates a "STATEMENT(S) NEEDED" message. It then invents and processes a sequence of statements which, if they had been included at that point in the program, would have placed the original source statement in a correct context. A message is given for each such statement invented. The original source statement is then processed in the newly created context.

By inventing statements the translator is not trying to patch up the program so that it will run correctly, it is simply trying to adjust the local context so that the original source statement and the statements which follow will be acceptable on a context basis. As in the case of context errors generated by ignoring a syntactically incorrect statement, such an adjustment of context frequently causes further context errors later on. This is called propagation of context errors.

12.3 Undetected Errors

LFP is ignorant of most details of FORTRAN syntax. Therefore most FORTRAN syntax errors will be detected by the FORTRAN compiler and not by LFP. In addition, there are two major classes of LFP errors which will be caught by the compiler and not the preprocessor.

The first class of undetected errors involves misspelled LFP keywords. A misspelled keyword will not be recognized by the preprocessor. The line on which it occurs will be assumed to be a FORTRAN statement and will be passed unaltered to the compiler which will undoubtably object to it. A common error, for example, is to spell UNTIL with two L's. Such statements are passed to the compiler, which then produces an error message. The fact that an intended control phrase was not recognized frequently causes a later context error since a level of indentation will not be triggered.

The second class of undetected errors involves unbalanced parentheses. (see also note 8 in Section 10.0). When scanning a parenthesized specification, the translator is looking for a matching right parenthesis. If the matching parenthesis is encountered before the end of the line the remainder of the line is scanned. If the remainder is blank or consists of a recognizable internal procedure reference, all is well. If neither of the above two cases hold, the remainder of the line is assumed (without checking) to be a simple FORTRAN statement which is passed to the Compiler. Quite often this assumption may be wrong. Thus the statement

"WHEN (X.LT.A(I)+Z)) X = 0"

is broken down into

```
keyword "WHEN"
specification "(X.LT.A(I)+Z)"
FORTRAN statement ")  X = 0"
```

Needless to say, the compiler will object to ") X = 0" as a statement.

Programmers on batch oriented systems have less difficulty with undetected errors due to the practice of running the program through both the preprocessor and the compiler each time a run is submitted. The compiler errors usually point out any errors undetected by the preprocessor.

Programmers on timesharing systems tend to have a bit more difficulty since an undetected error in one line may trigger a context error in a much later line. Noticing the context error, the programmer does not proceed with compilation and hence is not warned by the compiler of the genuine cause of the error. One indication of the true source of the error may be an indentation failure at the corresponding point in the listing.

LFP ERROR LIST

```
END statement is missing
Translator has used up allotted space for tables
CONDITIONAL or SELECT apparently missing
ELSE necessary to match FIN
FIN necessary to match line #
no control phrase for FIN to match
only TO and END are valid at this point
WHEN to match following ELSE
procedure already defined
procedure invoked but not defined
invalid character in statement label field
recognizable statement followed by garbage
left parenthesis does not follow keyword
missing a right parenthesis
valid procedure name does not follow TO
```

12.4 Control Card Errors

There are 4 control card error messages.

1. BAD INCLUDE FILENAME = XXXXXXXX YYYYYYYY

   This indicates that the file XXXXXXXX YYYYYYYY is not found. The include card is ignored.

   Reasons
   - misspelled filename or filetype
   - account not shared
   - specified file not blocked correctly

2. RECURSIVE INCLUDE DECKS NOT ALLOWED

   An include deck cannot include itself. The include card is ignored.

3. INCLUDES NESTED LEVEL GREATER THAN 40

4. BAD CONTROL CARD

   This catchall error indicates something was wrong with the control card.

   Possible reasons
   - misspelled control word
   - forgot control word
   - no argument present when one has needed
   - bad argument type
   - bad argument

   This error results in an ignored control card.

This error message is sent to the LFP listing and also to the terminal. The LFP line number (see 8.1.5 under LABEL control) is affixed to the error prior to printing at the terminal.

13.0 PROCEDURE FOR USAGE ON CMS - EXEC LPG

LFP at Lincoln Laboratory has been implemented on an Amdahl 470 system with the CP/CMS operating system.

The procedure which executes the LFP preprocessor is called LPG (see Fig. 13-1), which also will optionally perform macro substitution (see Section 15), generate a variable cross reference listing (Section 16) and compile the user's source program.

To gain access to the various EXECS and MODULES associated with LFP, type the following CMS command or put it in the PROFILE EXEC.

```
SHARE TOOLS
```

The LPG EXEC may then be executed by typing

```
LPG FN <FM> <(OPTIONS>
```

where

FN Filename of user's program with a filetype of LFP

FM Filemode of disk on which the user's program resides. The default is the "A" disk.

The following options are available.

COMP The generated FORTRAN is compiled by the G compiler (FORTGI).

NOCOMP The generated FORTRAN is not compiled.

DISK The listing output from the preprocessor is sent to disk (filename = FN, filetype = LFPLIST, filemode = A). The variable cross reference listing is sent to disk (filename = FN, filetype = XRLIST, filemode = A). These files contain carriage control and may be printed with the CMS commands:

```
PRINT FN LFPLIST (CC

PRINT FN XRLIST (CC
```

PR	The listing output from the preprocessor and the variable cross reference program are sent to the systems line printer.
NOPR	No listing output is generated.
XREF	Generate a variable cross reference table (see Section 16 and Figure 11.4).
NOXREF	Do not generate a cross reference table.
MACRO	Expand any macro definitions (see Section 14).
NOMACRO	Do not expand macros.
SAVE	The generated FORTRAN remains on the disk.
NOSAVE	The generated FORTRAN is erased subsequent to processing.

The symbols < > denote that the enclosed field is optional. All underlined options are the default. There are no abbreviations allowed for any of the above options.

The CMS command

```
LPG FN
```

is equivalent to

```
LPG  FN  A (COMP  NOMACRO  NOXREF  NOSAVE  DISK
```

The options list above may be printed at a terminal by typing

```
LPG ?
```

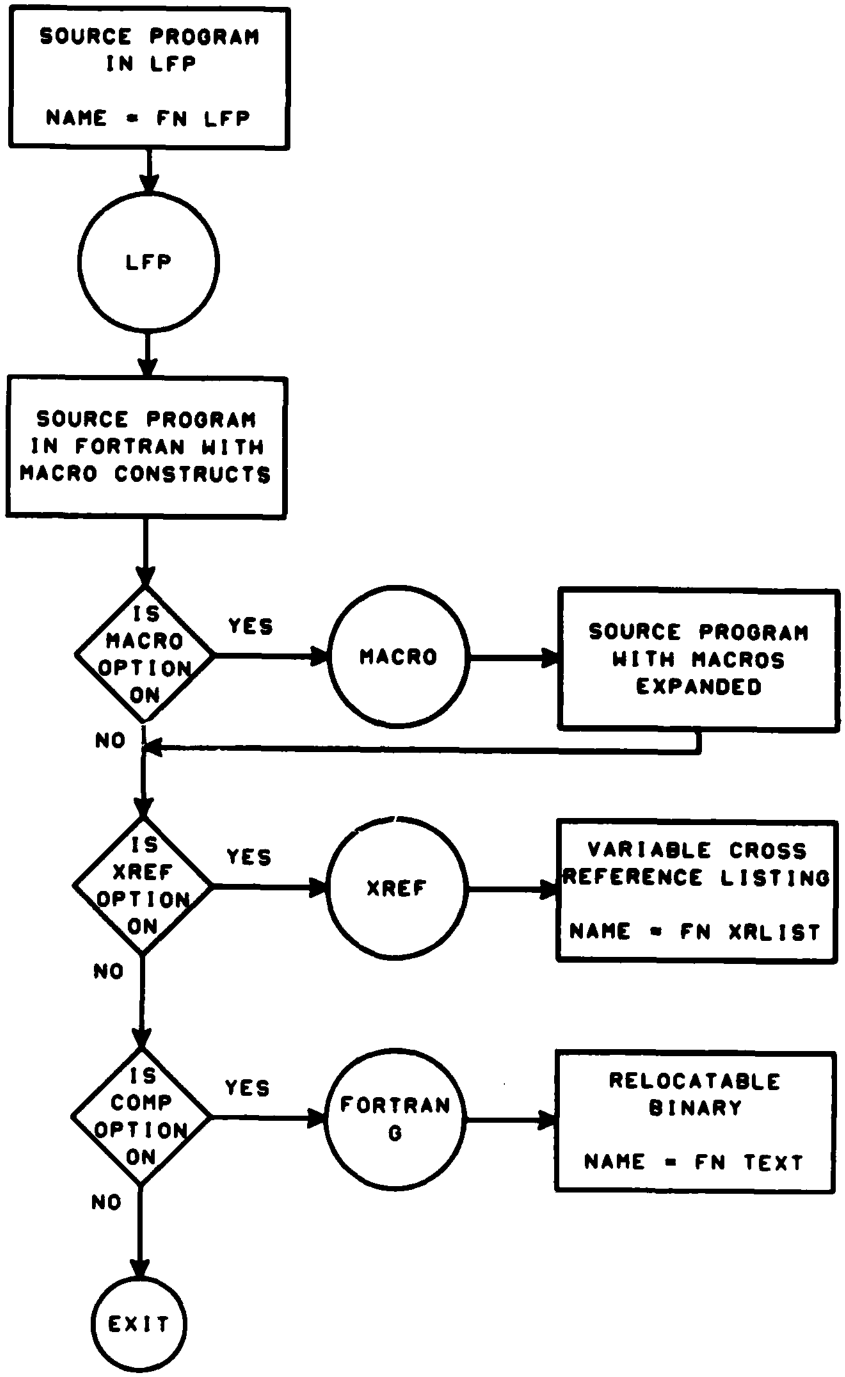


Fig. 13-1. LPG exec structure.

14.0 PROGRAMMERS' GUIDE TO LFP

14.1 Subroutine Description

LFP consists of a large main program (1800 lines) and approximately 30 subroutines (4200 lines). The purpose of each subroutine is listed below.

Name	Function
BLNKUT	Converts any leading zeroes in a character string to spaces
CATNUM	Convert a number to a character string
CATSTR	Concatenate 2 character strings
CATSUB	Concatenate a character string to a character substring
CHTYP	Classify a character to type
CLOSEF	Write # of diagnostics to terminal and listing and close files.
CONTOP	Controls the paging in the LFP listing
CPYSTR	Copy a character string
CPYSUB	Copy a character substring
GET	Processes control statements and inline comments
GETC	Get a specified character from a character string

GETCH	Get a specified character from a computer word
GETL	Read the next line of LFP from the mainstream or from an included file
GETTOK	Get a token
HASH	Compute Hash function
INIT	Initialization
LADJ	Remove any blanks in a source statement between column 7 and the first nonblank character
LFP	Main program
LITNUM	Convert a numeric character string to binary
NEWNO	Generate the next sequential statement label
OPENF	File initialization
PUT	Generate the FORTRAN, LFP listing and the error output
PUTC	Replace a character in a character string
PUTCH	Replace a character in a computer word
PUTL	Write 1 line to the FORTRAN, LFP listing or error file
PUTNUM	Put a 5 digit number at the beginning of a character string
STREQ	Logical character string equality compare

STRLT	Logical character string compare
TPAGE	Generate top of page header on the LFP listing
TRIM	Truncate trailing blanks from a character string

The named files contain the source code as listed below:

FILENAME	FILETYPE	CONTENTS
XFL6	LFP	Main program in LFP
XFL6	FORTRAN	Bootstrap of XFL6 LFP
XFL7	LFP	CATNUM
		CATSUB
		CHTYP
		CPYSTR
		CPYSUB
		GETC
		GETCH
		HASH
		PUTC
		PUTCH
		PUTNUM
		STREQ
		STRLT
		NEWNO
XFL7	FORTRAN	Bootstrap of XFL7 LFP

LFPUTIL	LFP	BLNKUT
		CLOSEF
		CONTOP
		GET
		GETL
		GETTOK
		INIT
		LADJ
		LITNUM
		OPENF
		PUT
		PUTL
		TPAGE
		TRIM
LFPUTIL	FORTRAN	Bootstrap of LFPUTIL LFP
LFPSUB	ASSEMBLE	CATSUB
		CATSTR
		CPYSTR
		CPYSUB
		CATNUM
		PUTNUM
		STREQ
		STRLT
		CHTYP
		GETCH

		PUTCH
		NEWNO
		HASH
LFPSUB2	ASSEMBLE	PUTC
		GETC
TRIM	ASSEMBLE	TRIM

Execs relating to LFP and their purposes are.

Name	Purpose
MAKELFP	Linkedit LFP (generate LFP MODULE)
LFPLIST	Describes LFP
LFP	Invokes LFP
LPG	Invokes LFP, Macro processor, variable cross reference processor and the Fortran G compiler.
MACRO	Invokes the Macro processor
XREF	Invokes the variable cross reference processor

14.2 Installation of a New LFP Version.

14.2.1 Bootstrapping LFP.

To bootstrap LFP, invoke the CMS commands:

```
FORTGI XFL6
FORTHX LFPUTIL (OPT(3)
ASSEMBLE LFPSUB2
ASSEMBLE LFPSUB
ASSEMBLE TRIM
LOAD XFL6  LFPSUB2  TRIM  LFPSUB  LFPUTIL  (NODUP)
GENMOD LFP
```

14.2.2 Installation parameter defaults

The user may wish to change certain default parameters in LFP depending on the computer system characteristics. It is recommended, however, that changes be made to the source version written in LFP--not the Fortran version and that the equivalent Fortran of each LFP version be archived. The following changes may be made:

14.2.2.1 Number of print lines/page

In subroutine INIT this is variable LNPPG which is currently set to 60.

14.2.2.2 Number of columns/page (width)

In Subroutine INIT this is variable LWIDTH which is currently set to 110. When subroutine GET processes a faulty WIDTH control statement LWIDTH is set to 110.

14.2.2.3 Default Heading

Subroutine INIT contains a data statement for the variable HDRDEF which defines the default heading.

14.2.2.4 LFP Version Number

Subroutine TPAGE contains a data statement for the variable VERSN to define the version number.

14.2.2.5 Default Control Character

Subroutine INIT defines the default control character. To change the default control character to an asterisk add the following code.

```
INTEGER STAR
:
STAR=92 ; HEX 5C
```

Replace the 2 lines

```
CALL PUTC (1, CNTRCH, PCNTC)
SVCNTC=PCNTC
```

with

```
CALL PUTC (1, CNTRCH, STAR)
SVCNTC=STAR
```

When subroutine GET processes a faulty CONTROL card, the percent sign is restored as a control character.

14.2.2.6 Default Statement Labelling/Counting

In subroutine INIT variable IBMMET controls the statement counting.

IBMMET=0	number all generated FORTRAN statements sequentially
IBMMET=1	number all generated FORTRAN statements sequentially except comments or continuation records.

Variable STNUML controls the line number at the left of the listing.

STNUML=0	Use the LFP line number (sequential from record to record)
STNUML=1	Use the FORTRAN statement number as determined by IBMMET

Variable STLABR controls the line number at the left of the listing:

STLABR=0	Use LFP Line number (5 columns)
STLABR=1	Use line tags (col 73-80) of input source record (8 columns)
STLABR=2	blank field

Subroutine GET redefines these fields if a faulty LABEL control statement is processed.

14.2.2.7 I/O Units

The I/O units are defined at the end of subroutine INIT. The unit numbers referenced by the FILEDEF statements in the exec LFP would also have to change.

14.2.3 Comment Delimiter

The comment delimiter is defined in subroutine INIT by the statement

CMTCH=SCLN

where SCLN is defined to be 94 Dec or 5E Hex.

When subroutine GET processes a faulty %COMMENT control statement, the semicolon is restored as the comment delimiter.

15.0 MACRO PROCESSOR

15.1 Description

The MACRO Processor, as described in chapter 8 of Software Tools [2] by Kernighan and Plauger, has been implemented on the Amdahl 470. This section briefly describes the function of the MACRO processor.

Macros are used to extend an existing language; tokens (alphanumeric character strings) may be defined and the subsequent usage of the token results in the replacement of the token by the macro definition.

The format of the macro definition for simple substitution is:

```
DEFINE(TOKEN,DEFINITION)
```

where DEFINE is a keyword which starts in column 1. There must be no imbedded blanks internal to DEFINE and no blanks between DEFINE and the left parenthesis.

Tokens must be alphanumeric (A-Z, 0-9, $) and contain at least 1 character. Examples of legitimate tokens are:

```
A     ALPHABET     8DOL     $NUM     5
$     $A1     $5     VERYVERYLONGTOKEN
```

An example of a macro definition is:

```
DEFINE(PI,3.145159265)
```

The macro reference for "PI" in the following statement:

```
CIRCUM = 2.0*PI*RADIUS
```

results in

```
CIRCUM = 2.0*3.14159265*RADIUS
```

Another example of a macro definition is:

```
DEFINE(RANGECELLS,50)
DEFINE(AZIMUTHCELLS,30)
```

The macro references for RANGECELLS and AZIMUTHCELLS in the statement

```
INTEGER*4  AMPTUD(RANGECELLS,AZIMUTHCELLS)
```

generates the code

```
INTEGER*4  AMPTUD(50,30)
```

Macros may also have arguments; the format of this type of macro definition is:

```
DEFINE(TOKEN,REPLACEMENT($1,$2,$3,...))
```

An example macro definition with arguments is:

```
DEFINE(ADD,$1+$2)
```

The reference to "ADD" in the statement

```
C = ADD(A,B)
```

results in

```
C = A+B
```

Macro ignores excess arguments during substitution, however, if an argument is not present the field $N is used.

Special care must be taken when the replacement string contains a macro reference, e.g., suppose a macro definition has been written for "A".

```
DEFINE(A,B)
```

and that later on the token "A" is to be replaced by the literal "C." The macro definition

DEFINE (A,C)

would result in all occurrences of A and B to be replaced by the literal "C".

To prevent this from happening MACRO allows escape characters; a left and right bracket enclosing a token will result in the token being copied to the Macro definition table without any substitutions.

Thus to properly redefine the token "A" by the string "C" the definition should be.

DEFINE([A],C)

Macro also contains the built in functions, INCR (token incrementing), SUBSTR (token substring) and IFELSE (conditional code generation). The reader is referred to Chap. 8 of Ref. 2 for an explanation.

Sample macro definitions and references are listed in Figure 15.1. The results after being processed by MACRO are listed in Figure 15.2.

```
FILE:  MTEST     INPUT     A1 F          80          54          6  2/18/81

*CASE 1 -------------------------PAGE 265
DEFINE(SKIPBL,WHILE($1($2) == BLANK | $1($2) == TAB)  $2  =$2  +1)
      SKIPBL(S)
      SKIPBL(S,I)
      SKIPBL(I,J,K)
*CASE 2 -------------------------PAGE 267
DEFINE(Y,X)
DEFINE(BUMP,$1=$1+1)
      BUMP(X)
      BUMP(Y)
*CASE 3 -------------------------PAGE 268
DEFINE(D,[DEFINE($1,$2)])
D(A,B)
      A
*CASE 4 -------------------------PAGE 274
DEFINE(XX,C$1B)
      XX(+)
      XX(-)
      XX(+,-)
      XX
      XX(*)
*CASE 5 -------------------------PAGE 274
DEFINE(PROC,[INTEGER FUNCTION $1  $2 DEFINE(PROCNAME,$1)])
     PROC(EQUAL,(STR1,STR2))
*CASE 6 -------------------------PAGE 276
DEFINE(MAXCARD,80)
DEFINE(MAXLINE,[INCR(MAXCARD)])
      MAXCARD
      MAXLINE
*CASE 7 -------------------------PAGE 276
DEFINE(STRNG,THIS IS A SAMPLESTRING)
      STRNG
      SUBSTR(STRNG,1,16)
      SUBSTR(STRNG,1)
      SUBSTR(STRNG,4,8)
      SUBSTR([SUBSTR(STRNG,11,6)],2,3)
      SUBSTR(SUBSTR(STRNG,11,6),2,3)
*CASE 8 -------------------------PAGE 276
DEFINE(COMPARE,[IFELSE($1,$2,YES,NO)])
      COMPARE(AA,BB)
DEFINE(CC,DD)
      COMPARE(CC,DD)
*CASE 9 ------------------------PAGE 280
DEFINE(LEN,[IFELSE($1,,0,[INCR(LEN(SUBSTR($1,2)))])])
      LEN(TEST)
DEFINE(SAM,SAMPLE STRING)
      LEN(SAM)
*CASE 10 -----------------------PAGE 280
*NOTE [LEN] IS DEFINED IN CASE 9
DEFINE(STRING,[INTEGER  $1(LEN(SUBSTR($2,2)))
STR($1,SUBSTR($2,2),0)       DATA $1(LEN(SUBSTR($2,2)))/EOS/])
                                DATA $1(INCR($3)))/[LET[SUBSTR($2,1,1)/
DEFINE(STR,[IFELSE($2,",,
[STR($1,SUBSTR($2,2),INCR($3))])])
      STRING(NAME,"TEXT")
```

Fig. 15-1. MACRO processor test examples.

```
FILE:    MTEST     OUTPUT    A1 F          80          41          5  2/18/81

*CASE 1 --------------------------PAGE 265
      WHILE(S($2) == BLANK | S($2) == TAB)  $2 =$2  +1
      WHILE(S(I) == BLANK | S(I) == TAB)  I =1  +1
      WHILE(I(J) == BLANK | I(J) == TAB)  J =J  +1
*CASE 2 --------------------------PAGE 267
      X=X+1
      X=X+1
*CASE 3 --------------------------PAGE 268
      B
*CASE 4 --------------------------PAGE 274
      C+B
      C-B
      C+B
      CB
      C*B
*CASE 5 --------------------------PAGE 274
      INTEGER FUNCTION EQUAL (STR1,STR2)
*CASE 6 --------------------------PAGE 276
      80
      81
*CASE 7 --------------------------PAGE 276
      THIS IS B SAMPLESTRING
      THIS IS B SAMPLE
      THIS IS B SAMPLESTRING
      S IS B S
      UBS
      AMP
*CASE 8 --------------------------PAGE 276
      NO
      YES
*CASE 9 --------------------------PAGE 280
      4
      13
*CASE 10 -------------------------PAGE 280
*NOTE LEN IS DEFINED IN CASE 9
      INTEGER NAME(5)
      DATA NAME(1)/LETT/
      DATA NAME(2)/LETE/
      DATA NAME(3)/LETX/
      DATA NAME(4)/LETT/
      DATA NAME(5)/EOS/
```

Fig. 15-2. Results of MACRO processor test examples.

15.2 Procedure for Using MACRO on CMS

The MACRO Processor on the Amdahl 470 is located on the TOOLS account. To invoke MACRO, type the following CMS commands

```
SHARE TOOLS

MACRO  FN1  FT1  FM1  FN2  FT2  FM2
```

where

(FN1, FT1, FM1) is the filename, filetype and filemode of the input file. MACRO expects as input a fixed blocked file with a record length of 80 bytes.

(FN2, FT2, FM2) is the filename, filetype and filemode of the output file.

The MACRO processor places on the terminal stack two parameters, &LEN1 and &LEN2. This results in columns 1 through &LEN1 being processed in the file FN1 FT1 FM1. The output file will only contain characters in columns 1 through &LEN2.

16.0 VARIABLE CROSS REFERENCE PROCESSOR - XREF

XREF is a variable cross reference processor for FORTRAN programs which is a major extension to the INDEX processor originally developed by H. M. Murphy.

Output from XREF includes an optional source listing, an alphabetical list of all symbols, a usage code which describes the symbol type and a list of statement numbers where the particular symbol was referenced. Figure 16-1 lists all cross reference usage codes. See Fig. 11-4 for sample output from XREF.

The XREF processor version 02.02 (XREF EXEC AND MODULE) are located on the account TOOLS. To use XREF, type the following CMS commands:

```
SHARE TOOLS
XREF  FN  <FM> <(OPTIONS>
```

where:

FN	The filename of the file to be cross referenced with a filetype of FORTRAN.
FM	Filemode of disk on which the file is resident. The default is the "A" disk.

The following options are available:

<u>DISK</u>	The printable output is written on the "A" disk with a filename = FN, filetype = XRLIST. The file may be printed with the CMS command:
	PRINT FN XRLIST (CC
PR	The output is sent to the systems line printer
NOPR	No printer output is generated
W=XXX	The listing width in columns is set to XXX. The default is 100 columns.

L=YYY	The number of print lines/page is set to YYY. The default value of L is 60 lines.
<u>S</u>	The FORTRAN source is listed along with the cross reference table.
NS	The FORTRAN source listing is not generated.

XREF will process ANSI FORTRAN statements but does have the following limitations:

1. Blank lines are not permitted (LFP removes them).

2. The first source statement should be either PROGRAM, FUNCTION OR SUBROUTINE. If the program statement is missing (as it should be for the Amdahl 470) the name MAIN is assigned.

Internal code	Usage code	Variable Usage
1		Blank line - ignore
2	=	Value changed by assignment statement
3	*	Line on which statement number is defined
4	AG	Subroutine Argument
5	CO	COMMON statement
6	CX	COMPLEX type statement
7	DA	DATA statement
8	DI	DIMENSION statement
9	DB	DOUBLE PRECISION statement
10	EQ	EQUIVALENCE statement
11	EX	EXTERNAL ENTRY name
12	IN	INTEGER specification statement
13	LG	LOGICAL specification statement
14	NM	NAMELIST specification statement
15	PR	PRINT statement
16	PU	PUNCH statement
17	RD	READ statement
18	RL	REAL specification statement
19	WR	WRITE specification statement
20	DO	DO Loop limit variable
21	CN	COMMON block name
22	RE	RETURN statement
23	ST	STOP statement
24	SN	Subroutine call
25	PA	PAUSE statement
26	ID	DO Loop Index variable
27	EN	END statement

Fig. 16-1. Cross reference symbol codes.

ACKNOWLEDGMENTS

LFP (Lincoln Fortran Preprocessor) is a major extension to the FLECS preprocessor which was originally developed by Terry Beyer at the University of Oregon.

We would like to thank Terry Beyer for his permission to use selected sections of the FLECS User Manual (notably sections 1, 2, 4, 5, 6 and 7) in the preparation of this report.

We would like to thank Paula Rygiel and Pam O'Connor for all the time that was spent in the preparation of the original report and thanks to Pam and Michelle Dalpe' for the work on this revised edition.

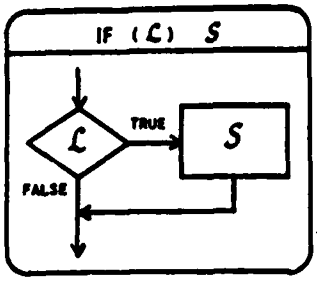
Bibliography

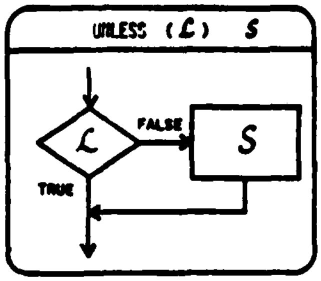
[1] Beyer, T., Flecs Users Manual (University of Oregon Edition)

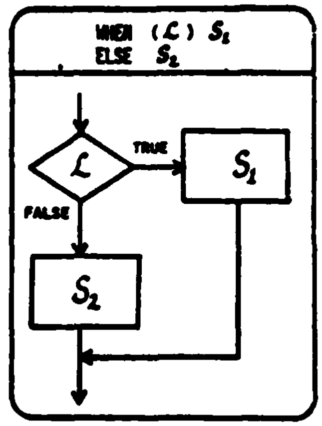
Contains a concise description of the Flecs extension of Fortran and of the details necessary to running a Flecs program on the PDP-10 or the IBM S/360 at Oregon.

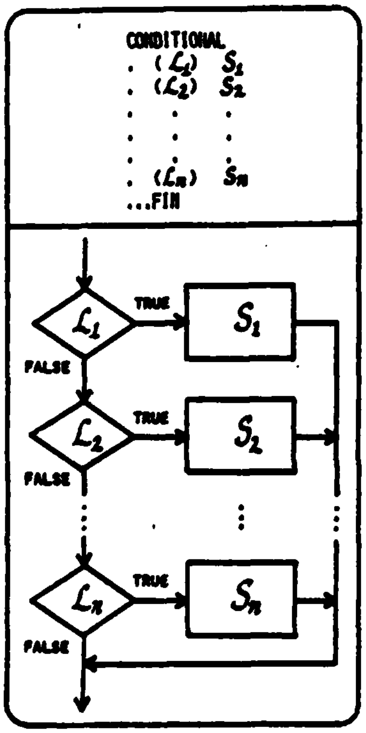
[2] Kernighan, B. W. and Plauger, P. J., Software Tools, Addison-Wesley Publishing Company, Reading, Mass (1976).

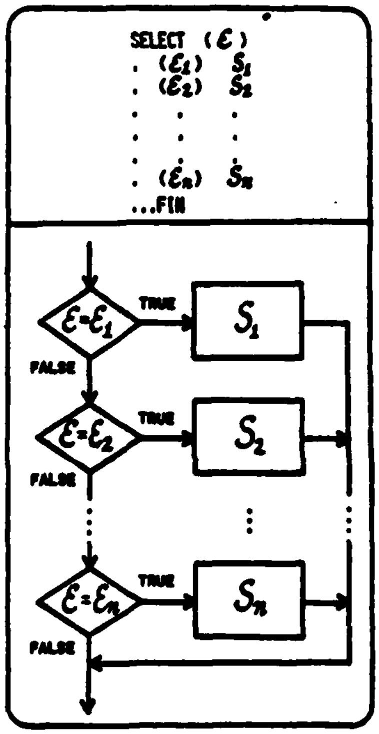
APPENDIX A - Control Structure Summary Sheet











CARRY-OUT-ACTION

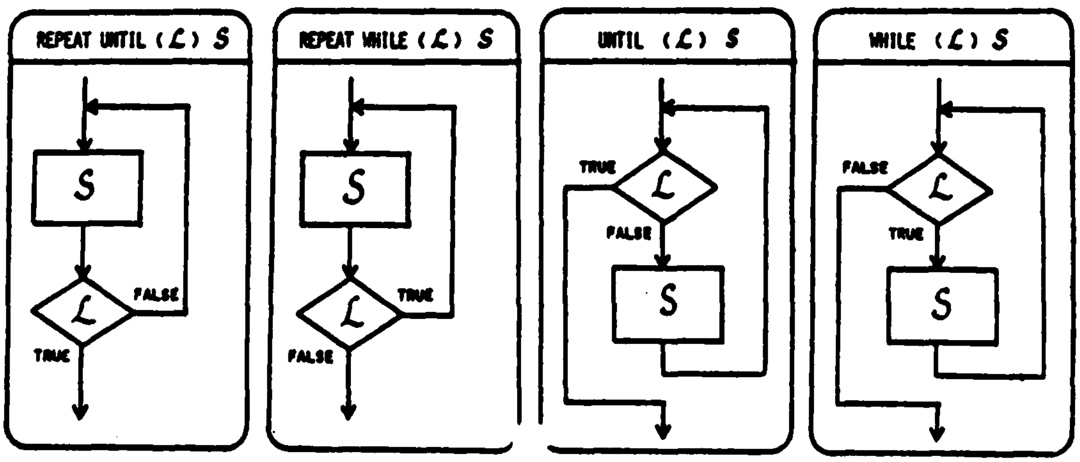
TO CARRY-OUT-ACTION S

DO ($\mathcal{I}$) S

NOTE: PLACE A RETURN, STOP, OR CALL EXIT STATEMENT AHEAD OF THE FIRST TO STATEMENT.

NOTE: OTHERWISE CAN BE USED AS A CATCHALL CONDITION OR EXPRESSION IN CONDITIONAL AND SELECT STATEMENTS.

LEGEND: $\mathcal{L}$ = LOGICAL EXPRESSION
$\mathcal{S}$ = STATEMENT(S)
$\mathcal{E}$ = EXPRESSION
$\mathcal{I}$ = DO SPECIFICATION



APPENDIX B - Control Statement Summary Sheet

Section

%COMment X — 8.1.1

Define the comment delimiter character X. Default is ;

%Control X — 8.1.2

Define the control character X. Default is %

%DS — 8.1.3

Double space the LFP listing

%Heading CHARACTER STRING — 8.1.4

Define the heading CHARACTER STRING to go at the top of each page of the LFP listing.

%Include FILENAME FILETYPE — 8.2.1

Include the contents of the file = FILENAME FILETYPE in the source file. The filetype must be LFP.

%INCExp N — 8.2.2

Controls the inclusion of a file on the INCLUDE card. if N is 0 the file is not included, if N is 1 the file is included. Default = 1.

%LABEL XYZ — 8.1.5

- X
  - 0 Modcomp FORTRAN line numbering
  - 1 Amdahl FORTRAN line numbering (default)
- Y
  - 0 LFP line numbers at left of listing
  - 1 FORTRAN line numbers at left of listing (default)
- Z
  - 0 LFP line numbers at right of listing
  - 1 Line tags at right of listing (default)
  - 2 blanks

%LADJ — 8.1.6

Left adjust the source to column 7, removing blanks.

%Line N — 8.1.7

Print N lines per page. Default = 60.

%List — 8.1.8

Print LFP listing. Default.

%NOLadj — 8.1.9

Do not left adjust source to column 7 (default).

%NOList — 8.1.10

Turn off LFP listing.

%Page N — 8.1.11

Eject a page if N=0 or N is missing.
Eject a page if there are less than N lines left on a page.

%SS — 8.1.12

Single space LFP listing (default).

%TS — 8.1.13

Triple space LFP listing.

%Width N — 8.1.14

Width of LFP output listing in characters.
Default = 110

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER ESD-TR-82-083	**2. GOVT ACCESSION NO.** A121004	**3. RECIPIENT'S CATALOG NUMBER**
4. TITLE *(and Subtitle)* LFP User's Manual (Lincoln Fortran Preprocessor) Version 02.01 for IBM/Amdahl Systems		**5. TYPE OF REPORT & PERIOD COVERED** Technical Report **6. PERFORMING ORG. REPORT NUMBER** Technical Report 623
7. AUTHOR(s) James H. Cosgrove and Edward T. Bayliss		**8. CONTRACT OR GRANT NUMBER(s)** F19628-80-C-0002
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lincoln Laboratory, M.I.T. P.O. Box 73 Lexington, MA 02173-0073		**10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS** Program Element No. 627[illegible]38A
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Electronics Command Ft. Monmouth, NJ 07703		**12. REPORT DATE** 1 September 1982 **13. NUMBER OF PAGES** 82
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* Electronic Systems Division Hanscom AFB, MA 01731		**15. SECURITY CLASS.** *(of this report)* Unclassified **15a. DECLASSIFICATION DOWNGRADING SCHEDULE**

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

structured FORTRAN LFP internal procedure

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

LFP (Lincoln Fortran Preprocessor) provides top-down control structures to FORTRAN and generates a self-documenting listing. LFP is compatible with existing FORTRAN and also permits an internal procedure capability.